NEW APPROACH FOR SOFTWARE PROCESSES REUSING BASED ON SOFTWARE ARCHITECTURES

Fadila Aoussat, Mohamed Ahmed-Nacer, Mourad Chabane Oussalah

NEW APPROACH FOR SOFTWARE PROCESSES REUSING BASED ON SOFTWARE ARCHITECTURES.

Fadila AOUSSAT,
Department of computer science, Saad Dahlab Blida University, BP270, Route Soumaa, Blida, Algeria.
A_zahoua@yahoo.fr

AND

Mohamed AHMED NACER,
Department of electronique and computer science, University of Sciences and Technology Houari Boumediène BP32,ElAlia, BabEzzoua, Algeria.
Anacer@cerist.dz

AND

Mourad OUSSALAH,
LINA Laboratoy, CNRS UMR 6241, 2, Rue de la Houssinière, BP 92208, 44322 University of Nantes, France.
Mouard.oussalah@univ-nantes.fr

ABSTRACT

This paper deals with reusing of software process models. Based on the insufficiencies of existing software process reusing approaches (limited reusability of the software process components), we propose a new approach that promotes a large reuse of existing proven software process models even not oriented components. Our approach is based on two steps: we use domain ontology to capitalize the software process knowledge and we handle the inferred knowledge as software process architecture. In this paper, we present a related works on this field and introduce the general outlines of our approach that is under validation.

Keywords: Domain ontology, inferring software process architecture. SPEM metamodel, automatic instantiation ontology, ADL for software processes.

1. INTRODUCTION

It is agreed that having high quality software process models has a direct impact on the software product quality. The main concern of the software process designer is to model software processes that highlight the characteristics of the project and that respect the local work tradition. The modeled software process must reflect the development reality, must be flexible and must "react" to the unexpected development events. Modeling high-quality software processes requires experience and a confirmed expertise. As a solution we explore the reusing of software processes that have been previously developed, tested, used and that have proven their efficiency.

Several approaches to modeling software processes based components have been proposed [1] [2] [3] [4] [16] [15] [7]. These approaches describe the concept of "Component Software Process" described as a fragment or a part of a software process, however while reusing components, each approach offers its own solution, addressing a particular aspect of modeling and executing software processes.

The major weakness of these approaches is that the developed software process components are specific to the environment; the use of the software process components is still limited to the environment itself. Even if the most existing approaches advance the same definition for a software process component, no consensus or metamodel describing the software process component characteristics is advanced. Thus, the concept of software process component On The Shelf "ready for use" has not yet appeared; so the immaturity and newness of this area is a logical justification for this work.

In the same context of reuse and at M2 of OMG four levels Architecture, the SPEM metamodel promotes reuse of software processes on a large scale. SPEM (Software and Systems Engineering Metamodel) [6], is a metamodel adopted by the OMG for modeling and executing software processes. Among other mechanisms of reusing, SPEM introduces the reused software processes based components by assigning a specific package: the "Method Plugin" package. SPEM introduces the architectural concepts (component, port, connector) more formally, however, reuse based components in SPEM is not yet mature, several problems must be taken into account, such as the heterogeneity of the terminology used for the same Work Product Port, or the lack of architectural concepts like software process configuration or software process architectural style.

Also, despite the repetitive nature of software processes (sequence of activities) a large number of software processes have emerged; each one using its own concepts, formalisms and terminologies to answer many specific needs. The established taxonomies and classifications [8] [9] [10] reflect the diversity of the proposed solutions. Based on the richness of the field in terms of concepts and experiences, as well as the limitation of existing approaches (software processes components weakly reusable, architectural abstraction not taken into account), we propose a new approach that has as main goal the reuse of existing proven software process models in term of knowledge. Also, we aim to offer a tool that gives a great flexibility to model new software processes, by being inspired on previous modeling positive experiences. Our solution is based on the use of a domain ontology which capitalizes this knowledge to allow an inference of new software process models.

By focusing on the architectural abstraction and addressing the software process as software architecture, we explore the automatic use of predefined structure of software processes such as life cycles or structure of predefined processes such as UP (Unified Process) to model new software processes. Therefore, the software processes that we develop are software processes based on software architectures. That's why we are interested on concepts derived from ADL's and their metamodel (few ADLs that are specific for software processes were developed but no metamodels that regroup architectural concepts for software processes were proposed). So, as first step, we will inspire from the existing ADL (Architecture Description Language) that are specific to software architecture.

The paper is organized as follows: Section 2 resumes the existing approaches for modeling processes based on software components. Section 3 presents the general outlines of our approach to modeling software process based software architecture. Our approach is based on the use of a domain ontology that contains software process knowledge thus Section 4 details the essential points for creating the ontology and discusses the encountered problems and the possible solutions. We conclude the paper summarizing the work and describing current works.

2. EXISTING APPROACHES FOR THE REUSE OF SOFTWARE-BASED METHODS OF COMPONENTS

In addition to the complexity when reusing software components, the presented approaches are facing more specific problems such as the rigidity of software process models, often depending on the modeling environment, and the diversity of the manipulated concepts [10]. Software process models are typically human oriented; interactions human/model has a central position, especially during the execution of software processes. The human element is the weak point of software processes; thus, adjustments are often made and must be integrated in the software process models. Also, software process models must be understood by their users, the vocabulary used to describe a "task" or "product" should be explicit and meaningful to the user, that's the raison of difficulties to reuse software processes models, particularly those coming from diverse sources.

We distinguish tow kind of approaches: approaches of the model level of the OMG modeling architecture, and approaches of the metamodel level.

2.1 Model level

2.1.1 Approaches based component concept

Several approaches to software process modeling based components have been developed. Each approach offers a particular solution, focusing on the concerns of its user, as the heterogeneity of languages process modeling software [1] [2], the heterogeneity of execution platforms [4], the distributed execution [3] or the conformity with SPEM meta-model.

The major weakness of these approaches is that the components developed are far from Component On The Shelf (COTS); in fact, these components are specific and their use is limited to their original environment. These systems typically operate so independently and do not reuse "external" components of their software processes.

The studied approaches (except APEL) use object-oriented languages for software process modeling; they implement their components as classes and use the object mechanisms (inheritance, instantiation ...) (Table 1-line -2 -).Unlike other environments, in APEL a component is a "product" component and not a "process" component; it is considered as a "support" for a local execution engine to execute a given part of the whole software process. Each software component has its own local process model [3]. APEL has been introduced into our study to have a general idea of the different concepts used on software process components.

Moreover, the notion of software process configurations and architectures in general, and logic configuration as abstract view in particular, has not beneficiate of much attention. The reflexion, often limited to the implementation level, is generally focused on the "content" of the component than the logic configuration and assembly. Consequently: 1) the architectural concepts (components, connectors, configuration, and architectural style), have been poorly exploited; most of the properties describing the component as a software component (dynamicity, non-functional properties etc ....) are not formally taken into account 2) the concepts of "connector" and "configuration" are not treated as first class entities: As the connector is considered as a function call, an event notification or an exchange message, its role is simply limited to communication between components [11], the used connectors have no independent existence and do not include additional mechanisms to facilitate and assist the interaction between components.

Component Characteristics	Environment RHODES	Framework OPC	PYNODE	ENDEAVORS	APEL
Creating period	Before the reuse(component repository)	During the reuse	During the reuse	Before the reuse, adapted during the execution.	Before the reuse
Processes Modeling Language(PML)	PBOOL+ Object oriented	Object oriented languages	Object oriented languages	ObjV based OOP LISP	Not specific language.
Heterogeneity	Homogeneous	Syntaxic	Syntaxic	Homogeneous	Syntaxic/Semantic
Assembling	Static	Dynamic and Incremental.	Dynamic and Incremental.	Static	No assembling
Metamodel	Some concepts	Use all concepts of the metamodel			
	SPEM metamodel	Basic elements (role/activity/ artifact)	Basic elements	Basic elements	Basic elements (activity, resource artifact)
Executing plateformes	Same platform	Same platform	Same platform	Multiple	Multiple
Identification.	Not assisted	Half assisted		Half assisted	No identification.
Reusability scope	Internal to the system				
Configuration management	No management (graphical representation of the assembly)				

Table1: Approach oriented object characteristics.

2.1.2 Approaches based architecture concept

Some approaches for software process modeling have been focused on architecture level:

Boehm [18] Argues that as software processes can be viewed as software, we can consider architecture styles for software processes. In [16] the treated software processes are evolution software processes. The proposed process architecture is software architecture for software process evolution. That consists on process components "evolution" and connectors specific to software process evolution. The architecture and components are described using language-specific trends: EPCDL (Evolution Process Component Description Language) and EPDL (Evolution Process Description Language). [17] Describes a method to model software processes based on object-oriented architecture. The method consists of "phases" and "concepts processes". The first phase defines the software process architecture with a large granularity, which is refined through other phases until obtaining the final software process model.

The solutions proposed by these approaches are either too specific to a type of a software process [16], too generic [17], or too general [18]. Moreover, these approaches don't allow to built predefined software process structures or software process configurations to generate software processes based software architecture. Our approach described in the next section manages these lacks.

2.2 Metamodel level

SPEM (Software and Systems Engineering Metamodel [6] is a metamodel adopted by the OMG, it describes a large range of software processes. Its organization into multiple packages offers not only several view points on the software processes (method view, structure view, reuse view ...) (Figure -1-), but also, facilitate the expansion and integration of new concepts.

SPEM supports different types of reuse: on one hand, while specifying "Process Behavior" package to capture external behavior of software process models that are not conform to SPEM metamodel, and on the other hand, while introducing reuse based on software process Components by providing another package: "the Method Plugin package". Through the concepts of "Process Component", "Process Component Use", "Work Product Ports" and "Work Product Connector" defined in the Method Plugin package, SPEM introduces more formally the notion of reuse component-based processes.

However, reusing components in SPEM faces several "recognized" problems that must be treated. The most important are the interconnection problems of components: heterogeneity of the terminology used for the port component "Work Product Port", the management of the number of ports per component creates difficulties for assembling components.

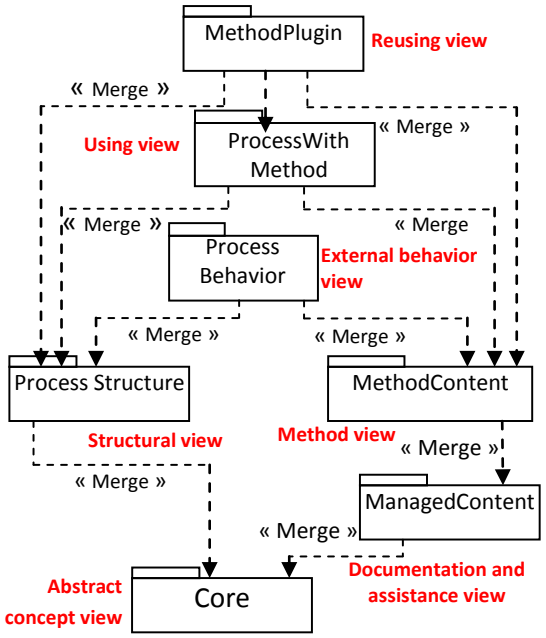

Figure 1 : SPEM metamodel structure [6]

The connectors defined in SPEM are implicit connectors. For instance, the "Work Product Connector" is a simple link between ports" Work Product Port ". These connectors don't play any role to facilitate the connection between software components, and their roles are limited to simply ensure communication between components; no mechanisms to facilitate connection have been integrated [11]. According to the SPEM cardinalities (Figure 2), a connector can connect multiple ports without any constraints, the concept of "connector role" is absent, and the correspondence port / connector is made manually; that creates multiple problems of connectivity between software process components. In addition, some properties for software connector (semantic, evolution ...) are not taken into account [12].

Like object-oriented approaches, the architectural abstraction (in other words, the manipulation of the software process as a set of software process components) has not been taken into account, and the notion of «software process configuration" remains unexplored even in SPEM (Figure-2-). The assembly of components in SPEM is done manually and often left to the judgment and experience of the software process developer. Consequently, the absence of software process configuration disallows speaking about software process architectural style.

SPEM is a UML profile; so it is clear that its gaps concerning some architectural concepts are inherited from the shortcoming of the UML2.0 metamodel about these concepts [12]. Indeed, UML allows specifying multiple fields through extension mechanisms; however, it is not the most suitable language for modeling software architectures. The UML2.0 metamodel lacks show clearly the shortcomings in the architectural concepts "Configuration" and "connector" in

SPEM.

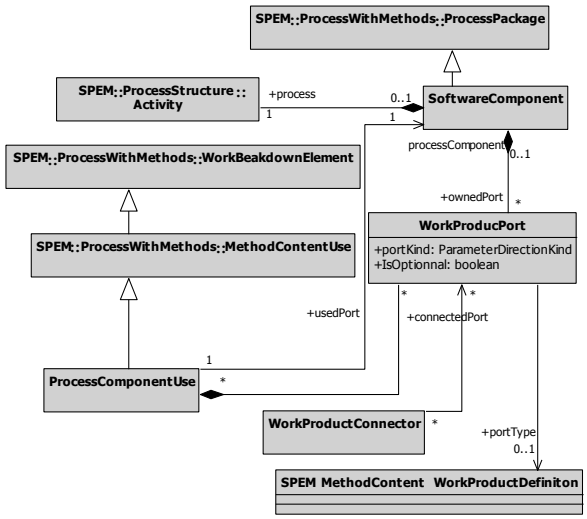

Figure 2: Architectural concepts for reuse based on components in SPEM [6]

3. OUR APPROACH

The approach for reusing software processes based on components that we offer is original because, it exploits every opportunity for reusing software processes to its extreme: firstly, while merging two research areas advocating the reuse at a large-scale (software architectures and ontologies) in the service of the software processes reuse, and secondly, exploiting all that was previously designed and used, as existing conceptualizations as SPEM metamodel, existing software process models, ATL module transformation model as UML2OWL **[13]**.

The main contribution of our approach lies in the fact that we model software processes as software architectures. We model the logical structure independently from the software process implementation. The architecture knowledge is inferred independently from the software process component knowledge and the result is saved as an XML file. The results will be described with a particular ADL. Also, we model the content of software process components regardless of the assembling structure, by developing pertinent queries that can infer a pertinent knowledge. The results will be used on the software process architecture deployment.

This separation is one of the characteristics of software architectures; that's allows us greater flexibility during the modeling process management and better control when modeling different kinds of software processes.

Unlike the discussed approaches, our approach covers both engineering "for reuse" and "by reuse".

- "For" reuse by providing an ontology which incorporates all "positive" experiences of previous software processes models.
- "By" reuse, allowing the inference of new software processes and their deployment.

3.1 Engineering for reuse

Using a domain ontology including most concepts of the software process field is the chosen solution to capitalize the software process knowledge. The ontology will form a support that contains the knowledge of this area, which will be reused regardless of their original environment. The instantiated ontology becomes a knowledge base, from which we can infer principally new software process models based on software architecture (Figure-3-). This step attempts to remedy the low reusability of software processes and to take advantages of the maturity of the field in terms of experiences and conceptualization. Our purpose is to infer using knowledge 1) issued from previous proven software processes models even not oriented components, 2) tailored to specific situations, and thus to have software process models with high-quality that meet the specific needs of software process developers.

To capture the experience of this area, the instantiation of this ontology must be based on existing proven software processes models. To capture such knowledge, a phase of reengineering is necessary and software process model analyzers must be developed for this purpose. The inconvenient, is that each modeling language for software process must have its corresponding analyzer to allow the capture of the knowledge. The instantiation from several software process models faces the problem of vocabulary heterogeneity; a relevant instantiation must identify distinctively each instance of the ontology. So, it is important to define a strategy to manage these synonymous instances (instances with their aliases). We think that a pertinent instantiation is the first step to achieve, in order to guarantee the success of our solution.

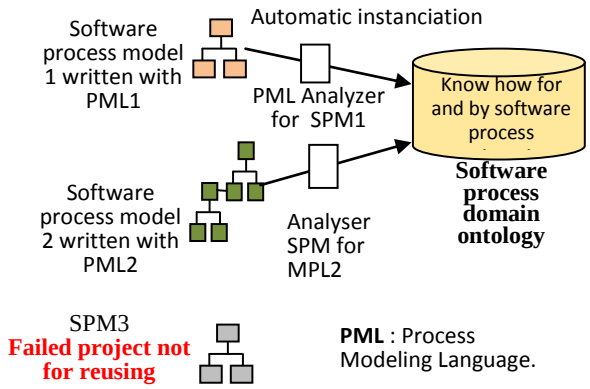

Figure 3: Capitalize proven knowledge in the software process engineering field.

3.2 Engineering by Reuse

Engineering by reuse is done by the inference of new software process models from the ontology knowledge. The query must consider the request of the process developer, and then infers the knowledge that matches developer requirements.

The query should allow the software process architecture inference, should identify software process components and their configuration (assembly). The assembly can be conform to a software process architectural style or not. The configuration of the software process at ontology level is a logical configuration. In fact, our approach separates the logical configuration from the operational configuration. In order to use and to reuse the inferred software process configurations, a support that model formally this knowledge as pure software architecture, in one hand, and tools that manipulate software process architectures, in other hand, are required. These tools must allow, firstly, to manage the software processes as logical software architectures without worrying about their implementation details, and secondly, to ensure their deployment.

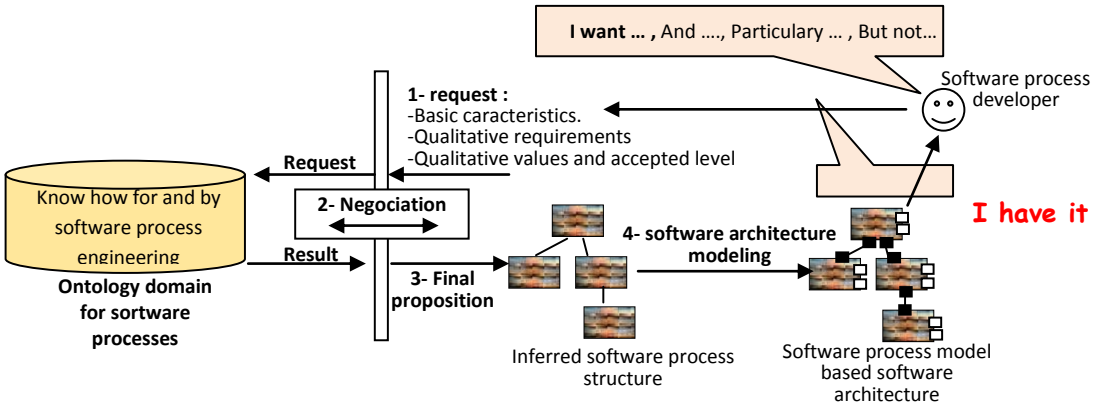

Figure 4: Software process modeling based on software architecture inferring.

To estimate the feasibility of this approach, it is necessary to evaluate the suitability of the architectural concepts to model the specific concepts of software process models.

Boehm highlighted the duality between the software product and the software process regarding software architectures [18]. He addressed a comparison between architectural concepts and software process concepts, concluding the interest to use software architectures styles to model software processes.

Thus, based on the identified concepts of existing approaches, an initial assessment gives us the possibility to underline that the software process models concepts (activity / product / activity sequence / process structure / life cycle) can be formally modeled as architectural concepts (component / port / connector / software configuration/ architectural style). However, it is clear that a deep study is needed to determinate formally the architectural representation of software processes.

4. DOMAIN ONTOLOGY FOR THE INFERENCE OF PROCESSES BASED SOFTWARE ARCHITECTURE

To capitalize the knowledge of software process engineering, our solution is based on a domain ontology. Our software process ontology must:

- Be consistent, unambiguous, and above all, commonly accepted,
- Be large and allow reasoning about different types of software processes.
- Represent explicitly software process architectures, including concepts and rules necessary for that purpose.
- Infer different software process configurations, respecting (or not) specific software process architectural styles.
- Infer software process components that match predefined constraints specific for software processes.
- Describe "assemblies" constraints for software process components and specific constraints for software process models.

To collect the concepts of our ontology, it is possible to exploit existing conceptualizations involving the basic concepts for modeling and executing software processes. Based on several established metamodels, the basic concepts of our ontology can then be obtained by projection of these meta-concepts. However, most of the existing metamodels are generic and represent the concepts of a particular environment; so, building an ontology from these metamodels is not adequate. That's why our work was oriented to the SPEM metamodel, that is more general, not specific to an environment and includes the concepts of several software process types.

4.1 Generation of the ontology by processing SPEM model

We developed a java application that generates our ontology "SPEMOntology" automatically from the SPEM metamodel, we use the models transformation language ATL [5]. ATL (Atlas Transformation Language) is a model transformation language based on the constraints languages OCL (Object constraints language) proposed by the OMG. It's defined to perform model transformations within the MDA (Model Driven Architecture).

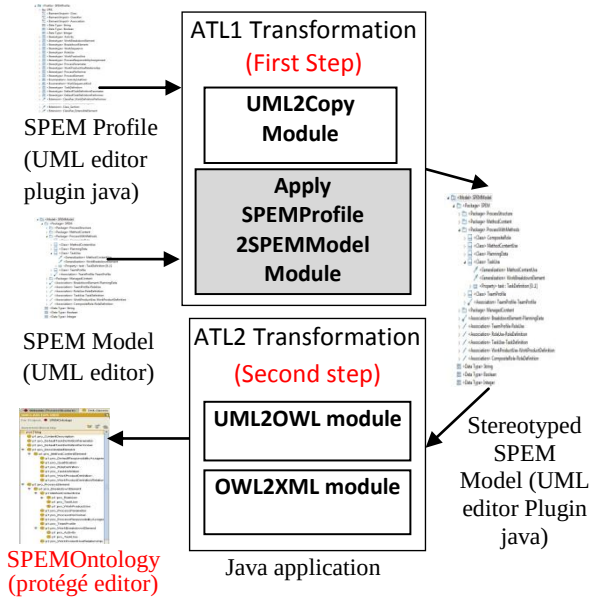

Figure 5: Steps of the SPEMOntology generating.

An ATL transformation is composed of ATL modules. For our transformation we use three existing transformation modules: UML2OWL OWL2XML and UML2Copy.

UML2OWL and OWL2XML are modules that provide rules for transforming a UML model into an OWL model. However, this transformation is not sufficient for our work, as it does not transform a "stereotyped" UML model conforms to a UML profile into an OWL model. The transformation UML2OWL does not contain transformation rules applied to profiles and their constituents (stereotypes, constraints and tagged values). In fact, the model is that we transform is a UML model (SPEM model) conforms to a UML profile (SPEM Profile). Thus, a previous ATL transformation is necessary (ATL1), this transformation must applied the SPEM Profile to the SPEM model, each element to its stereotype (UML class or association) to have stereotyped elements with constraints and tagged values.

Therefore, we define a new transformation (ATL1) which applies the profile SPEM to SPEM model. This transformation

is composed on two modules: 1) UML2COPY that copy not stereotyped elements from the source model to the target model, and 2) "AppplySPEMProfil2SPEMmodel" module that we have developed and that (as its name indicate) treats the stereotyped elements. Our java application executes the ATL1 transformation first; the target model "stereotyped spemModel" will constitute the source model of the next transformation (ATL2). The modules of the ATL2 transformation are executed in parallel, as they treat different elements (stereotyped and not stereotyped elements). Finally the SPEMOntology is generated and can be consulted with an ontology editor (figure 5).

5. CONCLUSION

This paper treats the reusing of software process models. We first identified the shortcomings of existing approaches: many approaches were proposed for modeling software processes based components, focusing however on a particular problem. Also, the defined software process component is low reusable and no consensus on the software process component characteristics is done. Also, as the reasoning on the architectural abstraction level is not being a priority; the representation of architectural concepts is insufficient.

Our paper introduces the general outlines of a new approach to modeling software processes based on software architecture. Our approach tempts to remedy the shortcomings of existing approaches (low reusability of software components, architectural concepts poorly exploited) and to exploit the reuse to its extreme: in fact, due to the rigidity and the dependency of software process to their development environment, high quality process models are developed and are not "re" exploited.

Our approach exploits the logic of software architectures: software processes are handled as software architectures. Thus, our approach derives its power from the separate handling of content, logical structure and deployment (that are inherited from the software architecture field). Our solution is very ambitious and aims to offer the opportunity to create new software process models from existing knowledge, by using a pertinent ontology, generated from a well accepted conceptualization (SPEM metamodel).

We believe that the exploitation of the architectural level of software processes will not only allows the effective reuse of knowledge in software process domain, but also, contributes significantly to facilitate and to resolve the modeling problems, the execution and the simulation of different software process structures:

- For traditional structures: by identifying architectural styles that are specific to software processes, based on software process and software life cycles [18].

- For specific structures: Such as dynamic, distributed, incremental and evolution software processes... allowing the management of architectural configurations for software process.

The validation of our proposition is under work. Multiple points remain to be developed: the extension of the ontology and the extension of SPEM with architectural concepts for software processes are the next targets of our work, we focus on definition of explicit connectors and styles specific to software process models.

REFERENCES

[1] Gary, K., Lindquist, T., Koehnemann, H., Derniame, J.-C. Component-based software process support. **13th IEEE International Conference on Automated Software Engineering.** 1998, Page(s):196- 199. DOI:10.1109/ASE.1998.732637.

[2] Avrilionis, D., Belkhatir, N., Cunin, P.-Y, A unified framework for software process enactment and improvement. **Fourth International Conference on the Software Process.** 1996, Page(s):102 – 111. DOI:10.1109/ICSP.1996.565028.

[3] Dami, S., Estublier J., Amiour, M. APEL: A Graphical Yet Executable Formalism for Process Modeling. **ASE'98, Automated Software Engineering.** Vol. 5, No. 1, January 1998. pp. 61-96. DOI :10.1023/A:1008658325298.

[4] Hitomi, A. S., Bolcer G. A., Taylor, R. N., Endeavors: A Process System Infrastructure. **ICSE'96, 19th International Conference in Software Engineering.** 1996. http://doi.ieeecomputersociety.org/10.1109/ICSE.1997.610424

[5] ATLAS group LINA & INRIA ATL: Atlas Transformation Language, ATL User Manual, version 0.7, Nantes, 2006.

[6] Object Management Group, Software & Systems Process Engineering Meta Model, v2.0 http://www.omg.org/cgi-bin/doc?Formal/2008-04-01.

[7] Belkhatir, N., Estublier, J., 1996, Supporting Reuse and Configuration for Large Scale Software Process Models. **ISPW'96,10th International Software Process Workshop.** DOI : 10.1109/ISPW.1996.65436.

[8] Zamli, K. Z., Process Modeling Languages: A literature review, **Malaysian Journal of Computer Science,** ISSN 0127-9084, Vol. 14 No.2, 2001.pp.26-37. http://ejum.fsktm.um.edu.my/ArticleInformation.aspx?ArticleID=100

[9] Silvia, T. Acuña, Xavier Ferré. Software Process Modelling. **World Multi conference on Systemics, Cybernetics and Informatics (SCI),** Orlando USA. 2001. DOI:10.1.1.23.5438

[10] Zamli, K. Z., Mat Isa. N. A., A survey and analysis of process modeling languages. **Malaysian Journal of Computer Science, ISSN 0127-9084** Vol.17 No. 2, 2004. pp. 68-89. http://ejum.fsktm.um.edu.my/ArticleInformation.aspx?ArticleID=310

[11] Nikunj, R. Mehta, Medvidovic, N., Understanding Software Connector Compatibilities Using a connector taxonomy, **SoDA '02, Proceedings of the First Workshop on Software Design and Architecture,** Bangalore, India. 2002. DOI :10.1.1.74.2629

[12] Medvidovic, N., Taylor R.N., A Classification and Comparison Framework for Software Architecture Description Languages. **IEEE Trans. Software Eng.** 2000. 26 (1): 70-93. DOI:10.1109/32.825767

[13] ATL transformation list, http://www.eclipse.org/m2m/atl/atlTransformations/

[14] Medvidovic, N., Rosenblum D. S., Redmiles D. F., Robbins, J. E., Modeling Software Architectures in the Unified Modeling Language. **ACM Transaction on Software Engineering and Methodology**, Vol.11, N°.1, January 2002. page 2-57. http://doi.acm.org/10.1145/504087.504088

[15] Kellner, M. I., Connecting Reusable Software Process Elements and Components. **ISPW'96, 10th International Software Process Workshop,** 1996**,** pages: 08-11.http:// doi.ieeecomputersociety.org/10.1109/ISPW.1996.654356

[16] Dai, F., Li, T., Zhao, N., Yu, Y., Huang B., Evolution Process Component Composition Based on Process Architecture. **International Symposium on Intelligent Information Technology Application Workshops.** Volume 00, 2008. Pages, DOI: 1097-1100. 10.1109/IITA.Workshops.2008.153

[17] Borsoi, B. T., Becerra J. L. R., A Method to Define an Object Oriented Software Process Architecture. **ASWEC'08, 19th Australian Conference on Software Engineering, Perth, WA,** 2008. pages: 650-655. http://doi.ieeecomputersociety.org/10.1109/ASWEC.2008.20

[18] Boehm, B. Wolf S., An Open Architecture for Software Process Asset Reuse. **ISPW'96, 10th International Software Process Workshop**, 1996. Pages:02-07. http://doi.ieeecomputersociety.org/10.1109/ISPW.1996.654354